

CLAIMS

What is claimed is:

1. A method for customization of a FPGA-based SoC, the method comprising:

selecting a system component used for customizing the FPGA-based SoC;

configuring said selected system component with parameters for use with the FPGA-based SoC;

propagating said parameters used to configure said selected system component to peer system components; and

configuring said peer system components using said propagated parameters during customization of the FPGA-based SoC.

2. The method according to claim 1, further comprising configuring the selected system component with parameters used to configure said peer system component.

3. The method according to claim 2, further comprising the step of propagating said parameter used to configure said peer system component to subsequently selected system components used to configure the FPGA-based SoC.

4. The method according to claim 1, wherein said selecting step further comprises the step of providing an option for selecting an implementation selected from the group consisting of a hardware implementation and a software implementation.

5. The method according to claim 1, wherein said step of selecting said system component further comprises selecting a system component from the group consisting of a hardware core

X-1002 US

and a software core.

6. The method according to claim 1, wherein the method further comprises the step of initializing only the selected system components that are utilized for customizing the FPGA-based SoC.

7. A machine readable storage having stored thereon, a computer program having a plurality of code sections, said code sections executable by a machine for causing the machine to perform the steps of:

selecting a system component used for customizing a FPGA-based SoC;

configuring said selected system component with parameters for use with said FPGA-based SoC;

propagating said parameters used to configure said selected system component to peer system components; and

configuring said peer system components using said propagated parameters during customization of said FPGA-based SoC.

8. The machine readable storage according to claim 7, further comprising sections of code for causing the machine to configure the selected system component with parameters used to configure said peer system component.

9. The machine readable storage according to claim 8, further comprising sections of code for causing the machine to propagate said parameter used to configure said peer system component to subsequently selected system components used to configure said FPGA-based SoC.

10. An interface for integrating hardware system component cores used for customizing a FPGA-based SoC, the interface

X-1002 US

comprising:

a slave connection circuitry communicatively interfaced to a processor bus;

a master connection circuitry communicatively interfaced to said processor bus; and

a multiplexer for selecting one of said slave connection circuitry and said master connection circuitry for providing communication between a processor bus and a selected hardware system component core used for customizing the FPGA-based SoC.

11. The interface for integrating hardware system component cores according to claim 10, wherein said selected hardware system component core is a proprietary customer specific hardware core.

12. The interface for integrating hardware system component cores according to claim 10, wherein said selected hardware system component core is a commercially available hardware core.

13. The interface for integrating hardware system component cores according to claim 10, further comprising a direct memory access (DMA) controller for providing direct access to a memory device.

14. The interface for integrating hardware system component cores according to claim 13, further comprising a write buffer and a read buffer, said write buffer and said read buffer providing temporary storage of I/O data from said memory device for said selected hardware system component core.

X-1002 US

15. The interface for integrating hardware system component cores according to claim 10, further comprising an interrupt controller coupled to said multiplexer, said interrupt controller for latching individual interrupt signals and providing a signal to a microprocessor indicating an interrupt condition.

16. A GUI for integrating system component cores during customization of a FPGA-based SoC, the GUI comprising:

a selection object for providing selection of a system component core for customizing the FPGA-based SoC;

a configuration object for configuring said selected system component core and peer system component; and

a display object for displaying parameters used by said configuration object to configure said selected system component core and peer system components.

17. The GUI according to claim 16, wherein said configuration object further comprises a parameter distribution object for distributing said parameters to subsequently selected system component cores and peer system components.

18. The GUI according to claim 16, wherein said selection object further comprises a dialog for selecting hardware system component cores and software system component cores.

**[0027]** During performance analysis, system analyzer 120 can have the capability to determine if system components are properly configured. For example, system analyzer 120 can identify a high-speed device that has not been configured with direct memory access (DMA). Since such a device can invariably cause a system conflict, system analyzer can consider it as a source of potential problem. System analyzer 120 can also determine whether there are too many devices residing on a bus based on the device count. For example, the system analyzer 120 can have the capability to determine whether there are too many high-speed devices on a low speed bus. In this case, the system analyzer 120 can indicate the possibility of errors and/or generate possible solutions. By tracking memory usage, the system analyzer 120 can have the capability to determine whether the code space assigned in the memory map is too large for the physical memory. System analyzer 120 can also be configured to track physical resource requirements for example, slice counts for IP blocks, and width and height of specifications of IP blocks. A GUI can provide a visual display of a resulting or representative floor plan to aid with tracking and management of physical resources.

**[0028]** Code generator 125 can include one or more GUIs having objects and/or dialogs that can facilitate generation of the code necessary for implementing the design of the FPGA-based embedded processor SoC or FPGA-based SoC. The code necessary for implementing the design of the FPGA-based SoC can be in a format such as the well known hardware description language (HDL). HDL is a language used to describe the functions of an electronic circuit for documentation, simulation and/or logic synthesis. Verilog and VHSIC Hardware Description Language (VHDL) are standardized HDLs which are well known by those skilled in the art. Verilog and VHDL can be used to design electronic

systems at the component, board and system level. They can facilitate the development of models at a very high level of abstraction. Other formats now known or to be discovered can also be used to represent the system model.

**[0029]** Depending on information generated by, for example, the software IP parameter customizer 115c, the code generator 125 can tailor "header files," which can be used to implement the software IP of the FPGA-based SoC. Moreover, depending on the selected software IP, processors, peripherals, operating system and device drivers, code generator 125 can produce a source code directory structure that can facilitate implementation of the software IP of the FPGA-based SoC. The code generator 125 can also generate the necessary "make files," which are files used to define the rules necessary for compiling and building the code used to implement the software IP of the FPGA-based SoC. The code generator 125 can be configured to generate information that can be used for debugging. The generated information can be in an ASCII format or other suitable format and can include information such as the memory map, the configuration ROM table and the peripheral ID map.

**[0030]** The system implementor 130 can include one or more GUIs that can have objects and/or dialogs that can facilitate implementation of the FPGA-based SoC design. Implementation of the design can include, but is not limited to, HDL simulation and synthesis, mapping of information generated by the code generator 125, placement, routing and bitstream generation. An integrated tool or separate tools can facilitate the implementation of the FPGA-based SoC design.

**[0031]** FIG. 2 depicts an exemplary topological view of the system model in accordance with the inventive arrangements. A GUI 200 can facilitate display of the topological view of the system model. GUI 200 can include a software development window 205, a simulation model window 210 and a core model

window 215. The software development window 205 can include one or more objects representing the various tools that can be used to create the code for the model system. Software development window 205 can include a compiler object 205a, an assembler object 205b, a linker object 205c, a converter or formatter object 205d and a SRAM object 205e. Software build tools such as compilers, assemblers, linkers, and converters are well known by those skilled in the art. Simulation model window 210 can include an OPB toolkit object 210a, a DCR toolkit object 210b, a PLB toolkit object 210c, an SRAM model object 210d and a SRAM model object 210e.

**[0032]** The core model window 215 can include objects representing the system components that can comprise the system model. Core model window 215 contains a 32-bit peripheral bus 215a, a DCR bus 215b, a 64-bit processor local bus 215c, a SDRAM controller 215d, a peripheral controller 215e, an arbiter 215f, a processor core 215g, a DMA controller 215h, a PLB/OPB bridge 215i and a serial port 215j. A high level view of the interconnections between the system components is shown in the core model window 215. For example, PLB/OPB bridge 215i provides interconnections between buses 215a, 215b and 215c. Bus 215c facilitates communication between peripherals including arbiter 215f, peripheral controller 215e, SDRAM controller 215d, DMA controller 215h and processor core 215g. Arrows between the objects in the core model window 215 and the simulation model window 210 can illustrate a relationship between corresponding objects in each model window.

**[0033]** FIG. 3 depicts a flow chart illustrating exemplary steps for integrating system component cores in accordance with the invention. Referring to FIG. 3, in step 380, system component #1 can be selected. System component #1 and any subsequently selected system component can include a hardware core or a software core. In step 382, system component #1

can be configured with parameters. In step 384, the system parameters can be propagated to make them available for subsequently selected system components that will utilize common parameters. In step 384, system component #2 can be selected. In step 388, system component #2 can be configured with parameters including previously propagated common parameters. In step 400, any new parameters that were used to configure system component #2 can be propagated to make them available for previously and subsequently selected system components that utilize similar parameters.

**[0034]** It should be recognized by those skilled in the art that the configuration and propagation of parameters can be static or dynamic. Notably, as system components are configured with new parameters, these new parameters are propagated and made available for configuring other system components. Importantly, these other system components can include previously configured hardware and software system components, as well as subsequently selected hardware and software system components. Advantageously, the propagation of system parameters can save on development time, since it can obviate any need to re-enter similar parameters which can be used to configure other selected system components.

**[0035]** FIG. 4 depicts an interface 480, for integrating software system component cores in accordance with the inventive arrangements. Referring to FIG. 4, there are shown an operating system layer 482, and operating system adaptation layer 484 and a system component layer 486. The operating system component layer 482 can facilitate management of resources for the software system components that are used to customize the FPGA-based SoC. The operating system layer 482 can host an operating system such as a RTOS.

**[0036]** The operating system adaptation layer 484 can facilitate communication between disparate system component drivers, for example 486a, 486b 486c, and the operating

system layer 482. The system component drivers 486a, 486b and 486c can be customer specific proprietary cores, each having a different communication interface. Since each of the component drivers 486a, 486b and 486c can have different proprietary interfaces, communication messages can be translated or converted to and from the proprietary formats to facilitate communication with the operating system layer 480. The operating system adaptation layer 484 can include a translator that can facilitate conversion to and from the proprietary formats, so that information can be communicated between the operating system layer 482 and the system component layer 486.

**[0037]** The system component layer 486 can include one or more system component drivers. Each of the system component drivers can be configured to handle the processing functions for a system component. For example, system component #1 driver can be configured to handle processing functions for system component #1. For illustrative purposes, system component #1 can represent serial device 215j. In this case, system component #1 driver 486a can be used to process data in an associated data buffer for serial device 215j. System component #1 driver 486 can include an interrupt handling routine that can be used to retrieve data pending in the associated data buffer for serial device 215j.

**[0038]** FIG. 5 depicts an exemplary hardware interface for integrating hardware system components in accordance with the invention. Referring to FIG. 5, there is shown an exemplary interface 500 that can facilitate integration of variously configured peripheral system components that can be utilized for configuring the FPGA-based SoC. The FPGA-based SoC can be configured to utilize dedicated transistors in the Silicon of the FPGA for implementing a peripheral interface. Alternatively, the FPGA-based SoC can be configured to utilize dedicated transistors in the logic fabric of a FPGA

for implementing a peripheral interface. Importantly, the choice of peripheral interfaces used for configuring the FPGA-based SoC can affect resource utilization of the FPGA-based SoC. For example, the resource utilization for a master-slave peripheral interface implementation can be markedly different from a slave only peripheral interface implementation. Advantageously, the invention can provide immediate feedback on system component and peripheral selection and implementation during customization of the FPGA-based SoC. Importantly, customization of the FPGA-based SoC can occur under resource constraints without the need to spend expensive development time and effort.

**[0039]** Exemplary interface 500 can include, but is not limited to, a multiplexer (MUX) 502, slave connection circuitry 506, master connection circuitry 508, direct memory access (DMA) controller 510, interrupt controller 504, address decoder 514, write buffer 516, and read buffer 518. The MUX 502 can facilitate selection of the slave connection circuitry 506 or the master connection circuitry, which can be used to connect a proprietary or customer specific or other hardware system component core 512 to processor bus 520. The interrupt controller 504 latches individual interrupt signals and provides an indication of an interrupt condition to a processor (not shown). DMA controller 510 can facilitate direct memory access to a storage device such as a random access memory (RAM). I/O data transferred to and from the system component core can be buffered in the write buffer 516 and the read buffer 518, which can both be selected by the MUX 502.

**[0040]** FIG. 6 depicts an exemplary GUI 350 that can facilitate integrating system component cores during customization of a FPGA-based SoC in accordance with the invention. GUI 350 can include a system component selection object 352, a parameter selection object 354 and a display

window 356. The system component selection object 352 can facilitate selection of system components that can be used to configure the FPGA-based SoC. System component selection object 352 can include a window having a plurality of system components that can readily be selected. In other words, the selection object may comprise a dialog for selecting hardware system component cores and software system component cores. For example, a radio button can be highlighted to select UART 16450 as a system component for configuring the FPGA-based SoC. Alternatively, one or more pull-down menus can be used to facilitate the selection of system components by their functionality. The system component selection object and the parameter selection objection essentially form a configuration object. The configuration object may further comprise a parameter distribution object for distributing the parameters to subsequently selected system component cores and peer system components.

**[0041]** Parameter selection object 354 can include one or more dialogs that can facilitate entry and or selection of parameters that can be used to configure system components. For example, system component UART 16550 can be configured with parameter 354a that indicates the use of interrupt request number 9 (IRQ9). The display window 356 can include a window having one or more objects representing allocation and use of resources.

**[0042]** Display window 356 can include one or more objects that can display the allocation of system resources as the system components are selected. A table object 362 can provide a condensed view of the allocation, usage and availability of the FPGA-based SoC resources. The table object 362 illustrates exemplary resources, which can include LUTs, D-FFs, Slices, BRAMS, I/Os, Memory and processor power. Specifically, the table object shows a breakdown of particular resources utilized by each device or system

component and also the total resources utilized by all the system components. The available resources can be computed based on the total utilized resources and the total device resources. For example, there are 122880 D-flip flops (D-FFs) available in the FPGA-based SoC. The table object can also show the processing power required by the system components. For example, the OPB arbiter utilizes 0.2 DMIPs, UART 16550 utilizes 1.4 DMIPs, and Ethernet 10/100M device utilizes 8.4 DMIPs. The total processing power required for use by the system components is 10 DMIPs. This leaves 290 DMIPs of the 300 DMIPs of the FPGA-based SoC processing power resources for use by other system components.

**[0043]** Customization is typically constrained by the availability of FPGA-based SoC resources. For example, the number of I/Os utilized by system components cannot exceed the number of I/Os of the FPGA-based SoC. Advantageously, the GUI 350 can provide real-time resource utilization feedback during the selection and configuration stages of the FPGA-based SoC development. By selecting a particular system component, the resources utilized by that system component can be immediately determined and displayed in the GUI 350. The resources can be itemized in order to simplify the tasks of selection and configuration of system components. In one aspect of the invention, each system component can have a corresponding application programming interface (API) the can provide an exact or estimated resource count.

**[0044]** Importantly, after all the system components have been selected, a system analysis and consistency check can be executed to ensure that the necessary system requirements and specification are met. For example, since processing delays can affect quality of service (QoS) in speech-based telephony applications, it is pertinent to ensure that processing delays caused by system components comply with the system specification and requirements. The system analyzer 120

(FIG. 1) can traverse the selected components and compute relevant resource allocation and usage statistics. These statistics can be used to optimize the allocation of system resources. System analyzer 120 can also provide functions, which can have the capability to determine system performance and power dissipation based on system component selection and parameter configuration. The GUI 350 can further include a message object 360 that can facilitate display of warning messages, error messages, inconsistency messages, and other types of messages and/or information. System component selection object 352 shows selected system components OPB arbiter, ethernet 10/100 and UART 16550.

**[0045]** In another aspect of the invention, a chip support package can be automatically created for the FPGA-based SoC. Typically, board support packages (BSP) can facilitate hardware and software customization. A BSP can include a circuit board and associated system and/or application software. The system and application software can include a collection of libraries, which typically isolate hardware functionality of the circuit board from its software functionality. For example, the BSP libraries can provide software functions that can be used for hardware initialization, interrupt handling, clock and timer management, and data structures for memory mapping and sizing. Nevertheless, a BSP usually correlates to a static design of a specific circuit board with specific components. A new circuit board with different components would then necessarily require a different BSP.

**[0046]** Advantageously, the FPGA-based SoC provides a more flexible approach than the BSP, by locating system components on a chip, in contrast to being on a separate circuit board. Rather than hard-coding the initialization of system components that reside on the circuit board of the BSP, the FPGA-based SoC can permit initialization of only those system

components that are utilized for customizing the FPGA-based SoC. This can drastically reduce initialization time and save on often precious memory. The code generator 125 and/or system implementor 130, can include a chip support package generator for generating a chip support package (CSP) or a board support package generator for generating a board support package (BSP) once the system components used to customize the FPGA-based SoC have been selected and configured. The code generator and/or system implementor can serve as a tool to automate the creation of a BSP based on a specific FPGA-based SoC and a specific operating system to be integrated with the hardware or software cores previously selected.

**[0047]** Advantageously, the ability to get real-time feedback and resource allocation can provide optimal resource allocation while configuring the system components used to customize the FPGA-based SoC. Notably, this can significantly reduce up-front development costs and non-recurring engineering costs and ultimately reduces the time to market. In light of the foregoing description of the invention, it should be recognized that the present invention can be realized in hardware, software, or a combination of hardware and software. A method and system for integrating cores during customization of an FPGA-based SoC according to the present invention can be realized in a centralized fashion in one computer system, or in a distributed fashion where different elements are spread across several interconnected computer systems. Any kind of computer system, or other apparatus adapted for carrying out the methods described herein, is suited. A typical combination of hardware and software could be a general purpose computer system with a computer program that, when being loaded and executed, controls the computer system such that it carries out the methods described herein.

**[0048]** The present invention can also be embedded in a computer program product, which comprises all the features enabling the implementation of the methods described herein, and which, when loaded in a computer system, is able to carry out these methods. Computer program or application in the present context means any expression, in any language, code or notation, of a set of instructions intended to cause a system having an information processing capability to perform a particular function either directly or after either or both of the following a) conversion to another language, code or notation; b) reproduction in a different material form.

**[0049]** Additionally, the description above is intended by way of example only and is not intended to limit the present invention in any way, except as set forth in the following claims.